NL000608
US

1c978 U.S. PTO
10/008195
11/08/01

Europäisches Patentamt

European Patent Office

Office européen des brevets

Bescheinigung

Certificate

Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. **Patent application No.** **Demande de brevet n°**

00203904.8

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

[illegible]

I.L.C. HATTEN-HECKMAN

DEN HAAG, DEN
THE HAGUE, 23/05/01
LA HAYE, LE

EPA/EPO/OEB Form 1014 - 02.91

Blatt 2 der Bescheinigung
Sheet 2 of the certificate
Page 2 de l'attestation

Anmeldung Nr.: / Application no.: / Demande n°:	00203904.8	Anmeldetag: / Date of filing: / Date de dépôt:	09/11/00

Anmelder:
Applicant(s):
Demandeur(s):
Koninklijke Philips Electronics N.V.
5621 BA Eindhoven
NETHERLANDS

Bezeichnung der Erfindung:
Title of the invention:
Titre de l'invention:
Best-case response times of periodic tasks

In Anspruch genommene Prioriät(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat: / State: / Pays:	Tag: / Date: / Date:	Aktenzeichen: / File no. / Numéro de dépôt:

Internationale Patentklassifikation:
International Patent classification:
Classification internationale des brevets:
/

Am Anmeldetag benannte Vertragstaaten:
Contracting states designated at date of filing: AT/BE/CH/CY/DE/DK/ES/FI/FR/GB/GR/IE/IT/LI/LU/MC/NL/PT/SE/TR
Etats contractants désignés lors du depôt:

Bemerkungen:
Remarks:
Remarques:

Best-Case Response Times of Periodic Tasks

Reinder J. Bril Liesbeth Steffens Wim F.J. Verhaegh

Philips Research Laboratories
Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
{reinder.bril,liesbeth.steffens,wim.verhaegh}@philips.com

In this paper we present a simple recursive equation to determine the best-case response times of periodic tasks under fixed-priority preemptive scheduling. The approach is of a similar nature as the one used to determine worst-case response times, in the sense that where a critical instant is considered to determine the latter, we base our analysis on an *optimal instant*, in which all higher priority tasks have a simultaneous release that coincides with the best-case completion of the task under consideration. The resulting recursive equation resembles much the one for worst-case response times, apart from a term -1 difference, and the fact that the best-case response times are approached from above. The resulting iterative procedure is illustrated by means of a small example. Finally, we discuss the effect of the best-case response times on completion jitter, as well as the effect of release jitter on the best-case response times.

Key words: response time, best case, periodic tasks, preemptive scheduling, fixed priority, jitter, real-time systems.

1 Introduction

One of the main performance issues in real-time computing systems is to determine whether or not a set of periodic tasks can be processed on a resource without exceeding their deadlines, using fixed-priority preemptive scheduling [2, 5]. To this end, constraints have been derived on workloads, to answer this question on an abstract level [6]. Another way is to have an exact worst-case analysis of response times of all tasks under arbitrary phasings [3] and compare this to their deadlines.

Given a set of n tasks $\tau_1, \tau_2, \ldots, \tau_n$, and given for each task τ_i a period T_i of activation and a worst-case computation time WC_i, the worst-case response time WR_i of a task τ_i is given by the smallest (positive) value that satisfies the following recursive equation.

$$WR_i = WC_i + \sum_{j \in hp(i)} \left\lceil \frac{WR_i}{T_j} \right\rceil WC_j$$

Here, $hp(i)$ denotes the set of tasks with a higher priority than τ_i. To calculate this worst-case response time, we can use the following iterative procedure.

$$\begin{aligned} WR_i(0) &= WC_i \\ WR_i(k+1) &= WC_i + \sum_{j \in hp(i)} \left\lceil \frac{WR_i(k)}{T_j} \right\rceil WC_j, \quad k = 0, 1, \ldots \end{aligned}$$

The procedure stops when the same value is found for two successive iterations k. Given these response times, we can conclude that the given set of tasks can be processed on the system if the worst-case response time of each task does not exceed its deadline, which we here assume to be equal to its period.

Although worst-case response times seem more interesting than best-case response times, there are sensible applications of the latter. For instance, if a certain task triggers a next one in a distributed multiprocessor system [1, 4], then the jitter (i.e. time variation) in the completion of the triggering task results in a release jitter of the following task, which in turn has an effect on the worst-case response times of other tasks. To this end, it is necessary to not only know the worst-case response time of the triggering task, but also its best-case response time, in order to have an as tight as possible bound on the jitter. Unfortunately, the lower bounds on the best-case response times given in [1, 4] are not tight. In this paper, we present a simple recursive equation to determine the best-case response times exactly.

The remainder of this paper is organized as follows. In Section 2 we present the notion of an optimal instant. Section 3 presents a recursive equation and an iterative procedure for the best-case response times, which is illustrated by means of an example in Section 4. Finally, in Section 5 we discuss jitter.

2 An optimal instant

Similarly to the notion of a critical instant [6], used to derive worst-case response times [3], the next theorem introduces the notion of an *optimal instant* to derive the best-case response time of a task τ_i, as depicted in Figure 1.

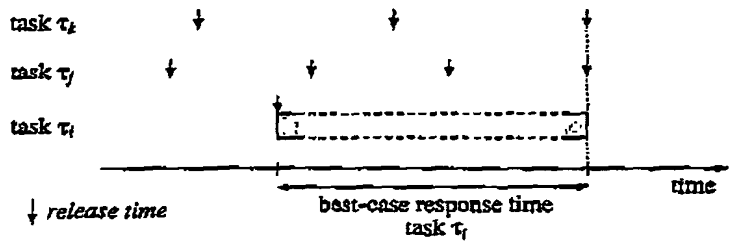

Figure 1. An optimal instant for task τ_i, where a release for all higher priority tasks (here τ_j and τ_k) coincide with the completion of an execution of τ_i.

Theorem 1. *An optimal instant for a task τ_i, i.e., an execution of it with best-case response time, occurs whenever its completion coincides with a simultaneous release for all higher priority tasks.*

Proof. The proof is based on a change argument, i.e., if we have an execution of τ_i with best-case response time but where not all higher priority tasks have a simultaneous release at its completion, we can slightly alter the phasings without increasing the response time of the execution of τ_i under consideration.

So consider such a situation, as depicted in Figure 2, where a higher priority task τ_j does not have a release coinciding with the completion of the execution of task τ_i under consideration. Now, as just before time t task τ_i is being processed, we know that between time t and r no activity of task τ_j occurs, since otherwise there would be work pending of a higher priority task just before time t. So, if we shift the releases of task τ_j by an amount $r-t$ to the left, no extra preemptions are shifted in from the right into the interval $[s,t]$ spanned by the execution of task τ_i. On the other hand, shifting the releases of task τ_j to the left may shift preemptions by task τ_j out of this interval

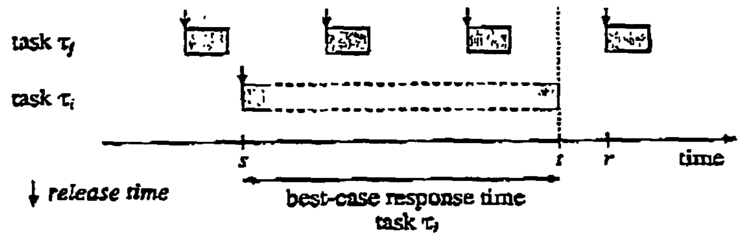

Figure 2. A situation not satisfying the definition of an optimal instant for task τ_i.

on the left. As a result, the amount of preemption of task τ_i in interval $[s,t]$ due to task τ_j, as well as due to other tasks τ_k, can only decrease or stay the same, and so we still have a best-case response time for the execution of task τ_i. In this way, the releases of all higher priority tasks can be shifted to the left to obtain an optimal instant. □

A consequence of Theorem 1 is that whereas the highest *condensation* of executions of higher priority tasks is found right *after* their critical instant, their highest *dilatation* is found right *before* it.

3 A recursive equation

Next, we derive a recursive equation for the best-case response times. Basically, we do this by looking back in time from an optimal instant.

Theorem 2. *The best-case response time BR_i of a task τ_i is given by the largest value that satisfies*

$$BR_i = BC_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{BR_i}{T_j} \right\rceil - 1 \right) BC_j, \tag{1}$$

where BC_j is the best-case computation time of a task τ_j. Furthermore, it can be found by the following iterative procedure, which stops when the same value is found for two successive iterations k.

$$\begin{aligned} BR_i(0) &= WR_i & (2) \\ BR_i(k+1) &= BC_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{BR_i(k)}{T_j} \right\rceil - 1 \right) BC_j, \quad k = 0, 1, \ldots & (3) \end{aligned}$$

Proof. Given an optimal instant for the response time of task τ_i, consider the preemptions of it by a higher priority task τ_j; see Figure 3. Now, as just after time s task τ_i is being processed, we

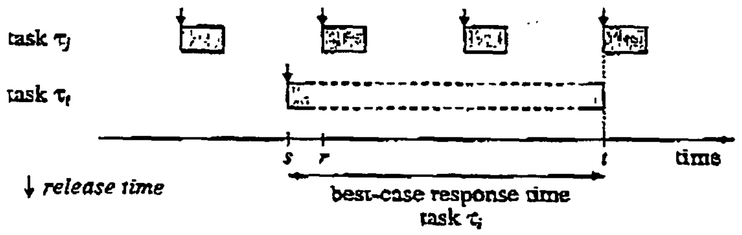

Figure 3. Preemptions of task τ_i by a higher priority task τ_j.

know that no activity of task τ_j takes place between s and r, since otherwise there would be work pending of a higher priority task at time s. As a result, only the releases of task τ_j *after* time s, i.e., the releases in the interval $[r,t)$, preempt the execution of task τ_i. The number of these preemptions is given by

$$\left\lceil \frac{BR_i}{T_j} \right\rceil - 1,$$

each of which gives a contribution BC_j to the best-case response time of task τ_i, resulting in the recursive equation (1). In order to have the tightest (i.e. highest) lower bound on the best-case response time, we should take the largest value satisfying this equation.

Next, if we have an upper bound $BR_i(k)$ on the best-case response time of task τ_i, and obviously $BR_i(0) = WR_i$ is one, then

$$\left\lceil \frac{BR_i(k)}{T_j} \right\rceil - 1$$

gives an upper bound on the number of preemptions by task τ_j in it, and thus $BR_i(k+1)$ as given in (3) gives a next upper bound on BR_i, which is at most equal to the previous upper bound. Upon termination, i.e., when $BR_i(k+1) = BR_i(k)$, we know that the gaps left by higher priority tasks in the interval $[s,t)$ in Figure 3 are just enough for the best-case computation time BC_i. This, together with the fact that task τ_i is being executed in the first part as well as the last part of the interval $[s,t)$, shows that the best-case response time cannot be smaller than the found value. □

In order to get a better initialization value in (2), we may replace the worst-case response times based on *worst-case computation* times by worst-case response times based on *best-case computation* times.

4 An example

To illustrate the iterative procedure for best-case response times, consider the example given in Table 1, where we assume the tasks to be given in order of decreasing priority. Figure 4 shows

task	period	computation time	worst-case response time	best-case response time
τ_1	10	3	3	3
τ_2	19	11	17	14
τ_3	56	5	56	22

Table 1. An example to illustrate the calculation of best-case response times. In this example, best-case computation times are taken equal to the worst-case computation times.

the successive iterations of the computation of the best-case response time of task τ_3, yielding an eventual value 22. As we can see in this figure, the initial upper bound $BR_3(0) = 56$, indicated by a dashed line, falls inside the period of execution 1 of τ_1 and the period of execution 1 of τ_2. As a result we can conclude that these executions do not preempt the execution of τ_3, so the preemptions can at most contain executions 2–6 of τ_1 and executions 2–3 of τ_2, resulting in a new upper bound of $BR_3(1) = 5+5\cdot3+2\cdot11 = 42$.

Next, in iteration (ii), we see that execution 2 of τ_1 also does not preempt the execution of τ_3, so the preemptions can at most contain executions 3–6 of τ_1 and executions 2–3 of τ_2, resulting in a new upper bound of $BR_3(2) = 5+4\cdot3+2\cdot11 = 39$.

This continues up to iteration (vi). There, we see that executions 5 and 6 of τ_1 and execution 3 of τ_2 preempt the best-case execution of τ_3, resulting in $BR_3(6) = 5+2\cdot3+1\cdot11 = 22 = BR_3(5)$,

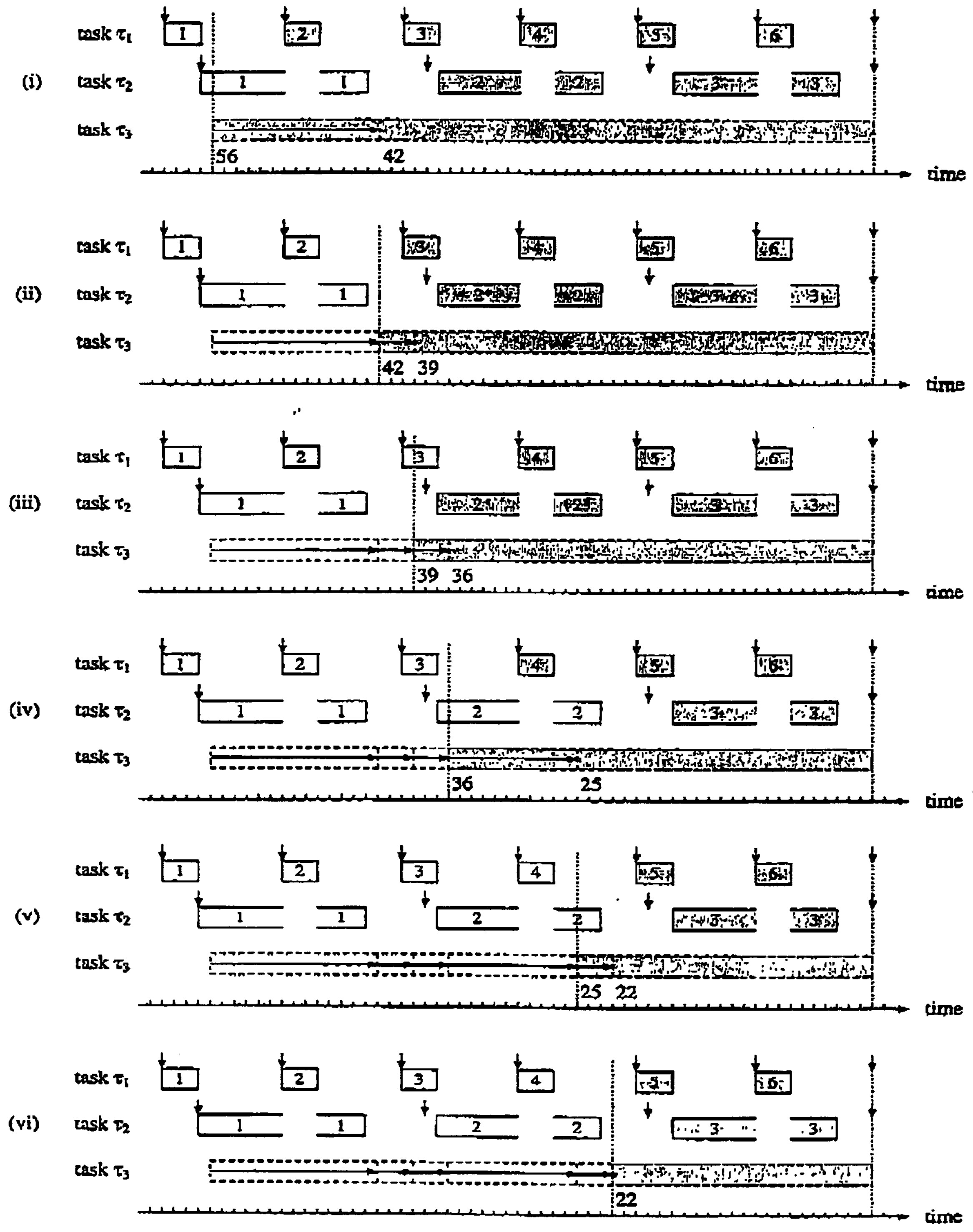

Figure 4. Iterations to determine the best-case response time of task τ_3. Executions are numbered for ease of reference.

and the procedure stops. So, the final best-case response time is 22, of which the execution and preemptions are shown in Figure 5. Using the techniques described in [1, 4] gives the same value

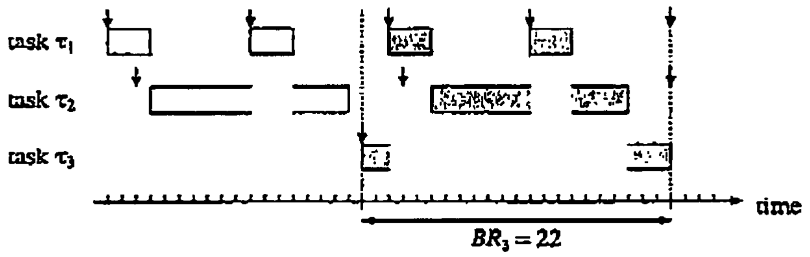

Figure 5. The eventual best-case response time of task τ_3.

for the best-case response times of τ_1 and τ_2, but only a lower bound of 5 for task τ_3, which is quite off the real best-case response time of 22.

5 Jitter

Given the worst-case and best-case response times of a task τ_i, we can calculate the completion jitter, i.e., the variation in the completion times of its executions as compared to a strict periodic pattern, with period T_i. If τ_i is released strictly periodic, the jitter is given by the difference between the worst-case and best-case response time, i.e.,

$$CJ_i = WR_i - BR_i.$$

Concerning completion jitter, we note that situations exist in which an execution of a task having best-case response time may directly be succeeded by an execution having worst-case response time, and vice versa. An example of this is given in Figure 6, where we have two tasks, τ_1 and τ_2, τ_1 having higher priority, with periods $T_1 = 8$, $T_2 = 12$ and computation times $WC_1 = BC_1 = WC_2 = BC_2 = 4$. For this example, the best-case and worst-case response times of τ_2 are $BR_2 = 4$ and $WR_2 = 8$, respectively, and the figure shows that they indeed can occur right after each other. The completion jitter of τ_2 equals $CJ_2 = 8 - 4 = 4$.

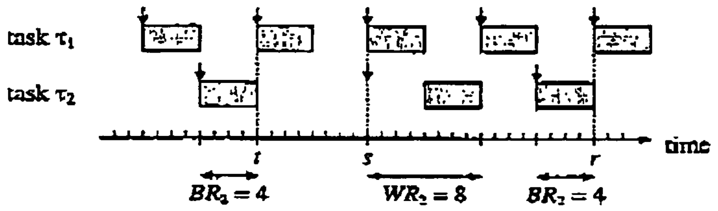

Figure 6. A situation in which a best-case execution of τ_2 is directly succeeded by a worst-case execution, which in turn is directly succeeded by a best-case execution. Note that times t and r indicate optimal instants for τ_2 and time s indicates a critical instant.

In case of release jitter, the analysis to derive worst-case and best-case response times as well as completion jitter is slightly altered. For worst-case response times, it changes as follows [2, 7]. Now, one has to consider the possibility that the jitter of a higher priority task τ_j occurs between the release at the critical instant and its next release, leading to a time between its first two releases that equals $T_j - RJ_j$, where RJ_j is the release jitter of task τ_j. Considering the worst-case number of preemptions over a time interval of length t then is similar to considering the preemptions of τ_j

as if it were strictly periodic, over a time interval of length $t + RJ_j$, and so the resulting worst-case response times are computed as follows.

$$\begin{aligned} WR_i(0) &= WC_i \\ WR_i(k+1) &= WC_i + \sum_{j \in hp(i)} \left\lceil \frac{WR_i(k) + RJ_j}{T_j} \right\rceil WC_j, \quad k = 0, 1, \ldots \end{aligned}$$

For best-case response times the effect of release jitter on the analysis is quite similar. Now we have to consider the possibility that the jitter of a higher priority task τ_j occurs between the release at an optimal instant and its previous release, leading to a time $T_j + RJ_j$ between these two releases. In turn, to determine the best-case number of preemptions of a task τ_j over a time interval of length t then is similar to considering the preemptions of τ_j as if it were strictly periodic, over a time interval of length $t - RJ_j$, and so the resulting best-case response times are computed as follows.

$$\begin{aligned} BR_i(0) &= WR_i \\ BR_i(k+1) &= BC_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{BR_i(k) - RJ_j}{T_j} \right\rceil - 1 \right)^+ BC_j, \quad k = 0, 1, \ldots \end{aligned}$$

Here, the notation x^+ stands for $\max\{x, 0\}$, which is used to indicate that the number of preemptions cannot be negative.

Given the worst-case and best-case response times of a task τ_i, as well as its release jitter, its completion jitter is now given by

$$CJ_i = RJ_i + WR_i - BR_i. \tag{4}$$

So, in order to take the effect of jitter into account in a distributed multiprocessor system with task dependencies [1, 4], where the completion of a task may trigger the release of a following task, we can use the following iterative procedure [1]. We start with an estimated release jitter $RJ_j = 0$ for all tasks τ_j, and do the above calculation of worst-case and best-case response times on each processor. Then we determine the completion jitter of each task, as given by (4). Next, we update the estimate of the release jitter of each task that is triggered by another task, by making it equal to the completion jitter of the triggering task. With these new estimates we then again determine worst-case and best-case response times, etc. During this process, the jitters and the worst-case response times increase, and the best-case response times decrease, causing again the jitters to increase etc., so eventually this process converges to a stable solution. Furthermore, this shows us that if we redetermine the worst-case and best-case response times for new estimates of the release jitter, we can use their final values of the previous iteration for initialization.

In general, the newly derived best-case response times lead to tighter bounds on completion jitter and thus also on release jitter of following tasks, in turn resulting in better worst-case response times of other tasks. A final remark that we would like to make is that we assumed arbitrary phasings in our analysis. If the dependencies of tasks are such that the phasings have a special structure, then the derived bounds on jitter etc. might still be pessimistic.

6 Conclusion

We have presented an iterative procedure to determine exact best-case response times of periodic tasks under preemptive fixed-priority scheduling and arbitrary phasings, based on a so-called optimal instant. Furthermore, we have shown how this can be used to determine jitter in multi-processor systems, where the newly derived best-case response times generally result in tighter bounds on jitter, in turn leading to better worst-case response times.

References

[1] J.C. Palencia Gutiérrez, J.J. Gutiérrez García, and M. González Harbour. Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems. In *Proceedings of the 10th EuroMicro Workshop on Real-Time Systems*, pages 35–44, 1998.

[2] M. Joseph, editor. *Real-Time Systems: Specification, Verification and Analysis*. Prentice Hall, 1996.

[3] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[4] T. Kim, H. Shin, and N. Chang. A jitter analysis for improved schedulability of distributed real-time tasks. *Journal of KISS Computer Systems and Theory*, 27(5):506–517, 2000.

[5] M.H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. González Harbour. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, 1993.

[6] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[7] K.W. Tindell, A. Burns, and A.J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.

A jitter analysis method using a novel formula for best-case response time

Reinder J. Bril and E. (Liesbeth) F.M. Steffens

September 26th, 2000

Real-time priority based systems often have strict jitter requirements. Jitter control decreases the effective processor utilisation, however. This note describes an improved analysis method, allowing less drastic jitter control measures, thus improving the effective processor utilisation.

Introduction

Real-time priority based systems often have strict jitter requirements. Jitter control decreases the effective processor utilisation, however. Especially for HVE (high-volume electronics) consumer devices, a high processor utilisation is very important for cost-effectiveness reasons. This note describes an improved analysis method, allowing less drastic jitter control measures, thus improving the effective processor utilisation.

Jitter is determined by computing a highest lower-bound (or best-case) and a lowest upper-bound (or worst-case) for the response time (see [1]). The computation time (optionally augmented with computations due to clock-interrupts) is typically taken as (rather pessimistic) highest lower-bound; stated in other words, the best-case response time is calculated under the assumption that no other tasks pre-empt (see [1], page 6.41, 6.48, 6.55, or 6.63). The value calculated by means of a well-known recursive equation for the worst-case response time is typically taken for the lowest upper-bound. The improved analysis method described in this note is based on a novel formula for calculating best-case response time. This novel formula takes pre-emptions of higher priority tasks into account, thereby improving the (rather pessimistic) highest lower-bound of the existing method.

Best-case and worst-case response time

A (fictitious) example of best-case and worst-case response times of a task is shown in Figure 7 The abbreviations in that figure have the following meaning:

- T – period;
- BR – Best-case Response time;
- BC – Best-case Computation time;
- WR – Worst-case Response time;
- WC – Worst-case Computation time;

Note that the task may be pre-empted by higher-priority tasks (giving rise to the "holes" in the computation). Best-case and worst-case response times are actually two sides of the same coin. Whereas worst-case provides an upper-bound for the latest moment in time that the actual work is finished, the best-case response time provides a lower-bound for the earliest moment in time that the actual work is finished.

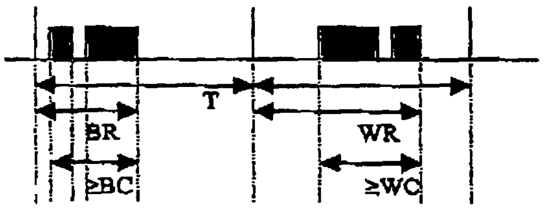

Figure 7 Best-case and worst-case response times

Recursive equation for worst-case response time

Neglecting blocking, the "standard" RMA (Rate Monotonic Analysis) recursive equation for determining the worst-case response time WR_i of a task τ_i is given by (see [1]):

$$WR_i = WC_i + \sum_{j=1}^{i-1} \lceil WR_i / T_j \rceil \times WC_j$$

The *smallest* value satisfying the equation is the solution for the worst-case response time.
This equation has (at least) three desirable properties:

1) it has a nice intuitive interpretation (the summation term represents the total pre-emption time by higher-priority tasks τ_j where $1 \leq j < i$);
2) there exists a simple iterative solution for the equation;
3) there is an easy-to-understand graphical representation using time lines and the underlying notion of a *critical instant*.

Recursive equation for best-case response time

Neglecting blocking, the recursive equation for determining the best-case response time BR_i of a task τ_i is given by:

$$BR_i = BC_i + \sum_{j=1}^{i-1} (\lceil BR_i / T_j \rceil - 1) \times BC_j$$

The *largest* value satisfying the equation is the solution for the worst-case response time.
This equation has (at least) two desirable properties:

1) it has a nice intuitive interpretation (the summation term represents the total pre-emption time by higher-priority tasks τ_j where $1 \leq j < i$);
2) there exists a simple iterative solution for the equation.

The best-case response time is based on the notion of *optimal instant*. The graphical representation using time lines is less instructive than for the worst-case response time, however.

The simple iterative solution for the equation is as follows:

1. The first approximation is obtained by taking the worst-case response time.
   $BR_i(0) = WC_i$
2. In subsequent approximations, less pre-emption effects are taken into account.

$$BR_i(n+1) = BC_i + \sum_{j=1}^{i-1} (\lceil BR_i(n) / T_j \rceil - 1) \times BC_j$$

3. The algorithm terminates when $BR_i(n+1) = BR_i(n)$.

Note that rather than taking *additional* pre-emption effects into account (as is done in a simple iterative solution for the recursive equation for the worst-case response time) *lesser* pre-emption effects are taken into account.

Concluding remarks

The proof of correctness for the recursive equation for the best-case response time and the underlying notion of *optimal instant* fall outside the scope of this note. The graphical representation using time lines is also not further explored in this note.

Best-case response times are also analysed in [2] and [3]. The approach described in [2] yields a more pessimistic value than the one found by the recursive equation presented in the note. For the time being, we did not receive (and hence didn't study) [3]. The abstract available on the www suggests that the paper describes *measurements* best-case response times rather than an analytical formula.

References

[1] Mark H. Klein, Thomas Ralya, Bill Pollak, Ray Obenza and Michael Gonzalez Harbour, *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer Academic Publishers, 5th printing, 1999.

[2] Taewoong Kin, Junghoon Lee, Heonshik Shin, and Naehyuck Chang, *Best Case Response Time Analysis for Improved Schedulability Analysis of Distributed Real-Time Tasks*, Journal of KISS Computer Systems and Theory (South Korea), 25(5): 506-517, May 2000.

[3] J.C.P. Gutierrez, J.J.G. Garcia, and M.G. Harbour, *Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems*, In: Proc. of the 10th Euromicro Workshop on Real Time Systems, 17-19 June, Berlin, pp. 35 – 44, 1998.

Real time priority-based systems often have strict jitter requirements. Jitter control decreases the effective processor utilization. Jitter occurs when a certain task triggers a next one in a distributed multiprocessor system. The jitter (i.e. time variation) in this triggering is important to know to determine worst-case response times of other tasks. Both worst-case and best-case response times of the triggering task must be determined to have a bound on the jitter. However, current best-case response times are not tight enough.

The invention determines the best-case response times exactly. The approach is of a similar nature as the one used to determiine worst-case response times, in the sense that where a critical instant is considered to determine the latter, the analysis is bases on an optimal instant, in which all higher priority tasks have a simultaneous request that coincides with the best-case completion of the task under considereation. In general, the analysis can be used to determine the best-case response times of periodic tasks under fixed-priority preemptive scheduling.

CLAIMS

1. A method of fixed priority preemptive scheduling a first task and a second task comprising steps of:
   - a first step of determining that the first task has a higher priority compared to the second task and that the second task has a lower priority compared to the first task,
   - a second step of determining a best-case response time of the lower priority task,
   - a third step of releasing the higher priority task at substantially a best-case completion time of the lower priority task.

09.11.2000

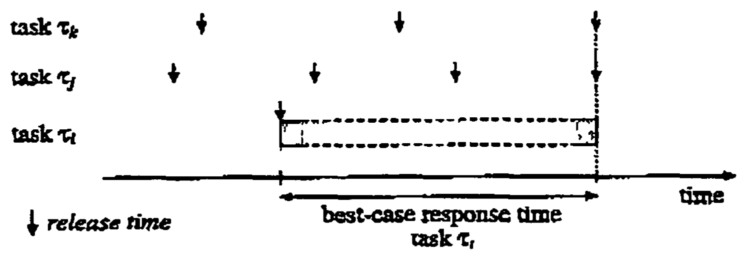

Figure 1

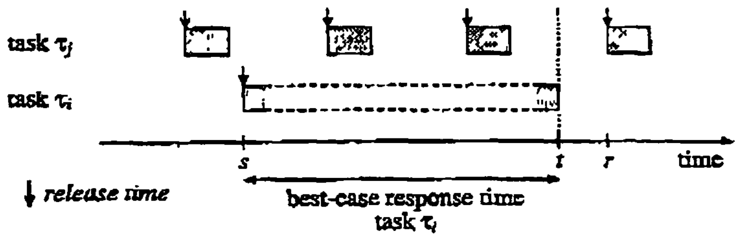
task τ_j
task τ_i
s
t
r
time
release time
best-case response time task τ_i

Figure 2.

09.11.2000

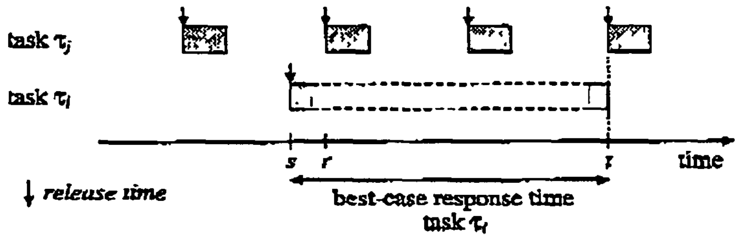

Figure 3

09.11.2000

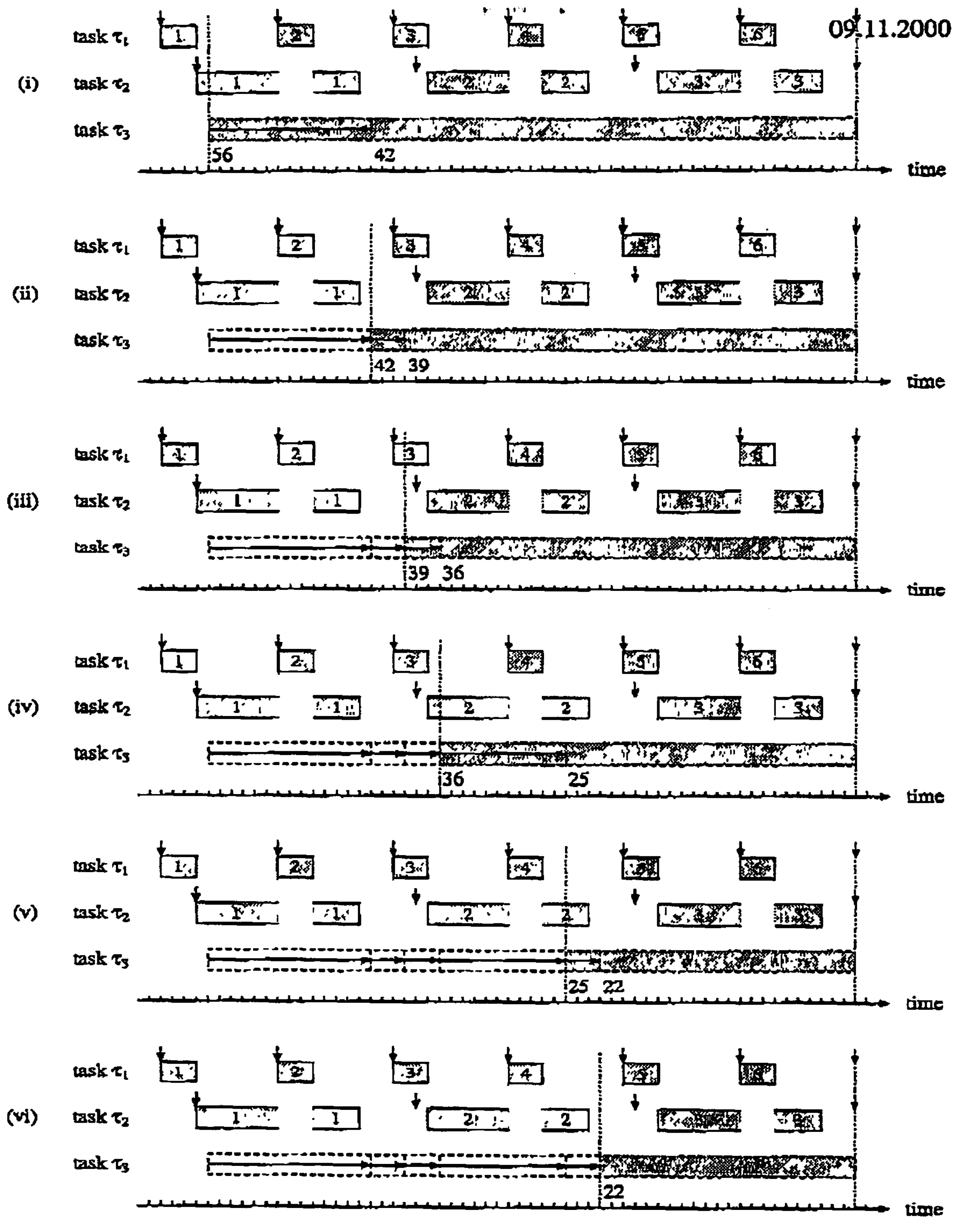

Figure 4.

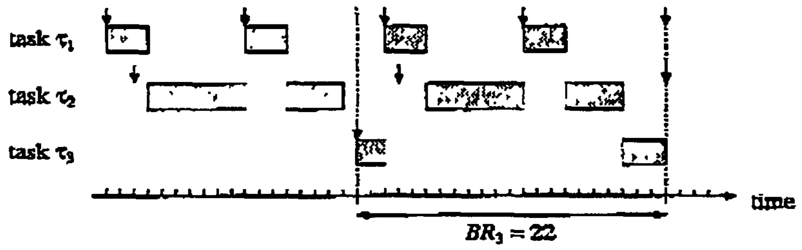

Figure 5.

09.11.2000

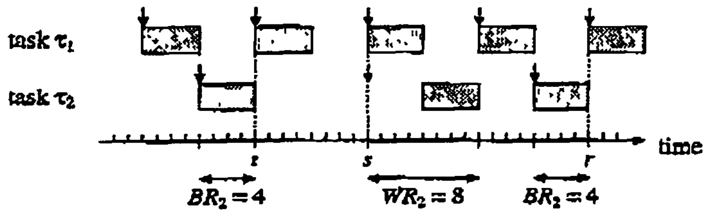

Figure 6

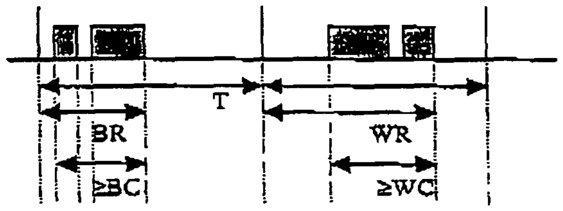

Figure 7